IBM

APL Programming Guide

Programming Conventions

APL Programming Guide

Programming Conventions

First Edition (July 1981)

APL PROGRAMMING GUIDE

PROGRAMMING CONVENTIONS FOR APL APPLICATION WORKSPACES

First Edition

Edited By

Timothy P. Holls

IBM Corporation

Poughkeepsie Information Systems

Department 944, Building 001-1

Poughkeepsie, New York

ABSTRACT

This Guide summarizes conventions and guidelines for preparation of APL programs.

Last Updated 3/03/81

PROGRAMMING CONVENTIONS FOR APL APPLICATION WORKSPACES

PREFACE

This document summarizes conventions and guidelines which are appropriate for APL Application Workspaces. APL Application Workspaces are debugged APL programs (sets of defined functions) which are regularly executed to produce results needed for support of development, manufacturing, and business operations.

These conventions represent generally accepted, good practices for preparation of APL programs. However, they cannot all be rigidly required for every APL Application Workspace. Exceptions to these conventions may be appropriate depending on the user and application environments. Nevertheless, this document should be reviewed for all applicable conventions, and any exceptions should be made on the basis of recognized need. While there are alternatives to some of these conventions, these specific conventions are offered as a useful set to those programmers and installations who have not yet adopted their own.

The use of these conventions will promote a common style in APL program code. This will contribute to ease of understanding and allow easier transfer of maintenance responsibility. These conventions also form a basis for guiding new APL programmers into good programming habits.

The section on "Local System Conventions" should be replaced with special considerations and requirements relevant to the particular installation and the APL system in use.

These conventions were compiled from a number of reports and papers and the contributions of reviewers. Special thanks are due S. Eusebi, J. Ever, C. Fair, L. Lewis, B. Martin, Y. Morita, P. Nicholson, R. Polivka, J. West, and C. Wilkes for material and comments.

PROGRAMMING CONVENTIONS FOR APL APPLICATION WORKSPACES

TABLE OF CONTENTS

PROGRAMMING CONVENTIONS FOR APL APPLICATION WORKSPACES

I. DOCUMENTATION

A. WORKSPACE DOCUMENTATION

1. Workspaces must contain a *DESCRIBE* variable or function which provides documentation about the workspace.

2. The *DESCRIBE* may refer to hardcopy documentation if it exists. In this case, *DESCRIBE* may contain just a brief summary of the nature of the application.

3. If the *DESCRIBE* is the main documentation, it must give sufficient information to understand how to use the workspace. All user-invoked functions must be itemized with explanations.

4. If the *DESCRIBE* is very large, it may be kept in a separate workspace. In this case, the *DESCRIBE* in the production workspace should simply point to the separate workspace.

5. If a workspace consists of independent functions, documentation for each function may be included in a separate function or variable named with the suffix *---HOW* appended to the function name.

6. Auxiliary workspaces may contain a *DESCRIBE* which refers to the main workspace.

7. Latent expressions (⎕*LX*) should not invoke a lengthy *DESCRIBE*. If it is important, they should merely direct the user to the main documentation.

8. Related workspaces should be named using the same prefix name, e.g., *PLOT* and *PLOTUNLK* (not *UNLKPLOT*). This will allow their names to appear together in a)*LIB* listing.

9. A variable named *CONTROL* must contain the following information (alternative name: *AUTHOR*):

   Author/maintainer, phone, dept/bldg, loc

10. A variable named *WSID* must contain the library number and name of the workspace and the date last updated.

I. DOCUMENTATION (CONTD)

B. PROGRAM DOCUMENTATION

1. Every function of an application must have a commented version stored on the system, which is maintained as the source copy. Comments may be deleted and/or the function locked to produce a separate execution-copy of the application.

2. Every function must contain the following comments as its first lines:

   Line 1: Concise explanation of what the function does. This line should start with ⍝⍝.

   Line 2: Level-number, date of last revision, programmer name.

   Subsequent lines: Descriptions of
   (1) Arguments.
   (2) Explicit result.
   (3) What functions are called.
   (4) What variables global to this function are used, changed, or created.
   (5) What files are processed.
   If necessary, this can refer to separate documentation in an "application notebook".

3. Comments must be included in all functions to explain, as necessary, the steps in the processing and WHY they are done. Special notes should be included about unusual conditions, dependencies, or side effects.

4. Comments must be on separate lines, starting with ⍝, preceding the statements which they explain. This is to allow easy deletion for an execution copy, if necessary. There should be an minimum average of 1 comment line per 3 lines of code (25 percent comments).

5. Comments must be kept up-to-date and accurate.

I. DOCUMENTATION (CONTD)

C. APPLICATION MAINTENANCE DOCUMENTATION

The following documentation must be produced and maintained in a note binder for each application workspace. These items should be produced using automated utility programs where available.

1. Listings of the commented version of every function.

2. Listings of the characteristics and values of all global variables stored in the workspace or on a file.

3. A description of the structure and record formats of all files accessed or created on secondary storage, the auxiliary processors required, and any APL system environment dependencies.

4. A summary of the environment of each workspace, displayed using the system commands:

```
)LOAD wsname  (to show name, date-saved, and
                ⎕LX messages)
)FNS
)VARS  (should have erased temp vars, init globals)
)GRPS
)GRP grpname (for each group)
)SYMBOLS  (should have been cleaned up via a )COPY)
⎕WA
⎕PW
⎕IO
⎕LX
)SI  (should be empty)
Other system variables, if not the same as in a
 CLEAR WS.
```

5. An alphabetized summary listing of all functions and what they do.

6. An indented listing of the function-calling structure, starting with each user-invoked function.

7. Summary tables of global variables (used in what functions), function calls (from what functions), and workspace statistics (number of: functions, names, lines of code).

II. HUMAN FACTORS

The level of user ability and experience with computers must be considered when designing an APL application. The following are considerations regarding understandability and user convenience.

1. User-invoked functions should have short names (not more than 8 chars) that are meaningful and mnemonic. They should be niladic (no arguments). User inputs should be requested via prompting messages. This reduces the need to remember the syntax of commands.

2. All prompting for user inputs should be done near the beginning of execution, if possible, especially when there is lengthy program execution. Prompting should be combined into one function in the program. This will allow the processing code to be used independently of any prompting.

3. Lengthy processing should be split up into steps or phases. Do not make the user repeat any more than necessary to re-do or change the results. Display "progress messages" or transaction/record counts at key points in program execution to indicate the phase of processing which has been reached. This will indicate to the user that execution is continuing or has completed.

4. Prompting messages should be clear to the user, but short. Long explanatory messages should be suppressible or have an alternate short form for experienced users. Messages should not wrap-around on the terminal screen. A width of 64 characters fits on the 51xx series, 76-80 on the 327x.

5. Extensive message text or menus should fill the terminal screen width, occupying as few lines vertically as possible. The complete text should fit on the screen at one time. If necessary, the terminal screen should be cleared first, under program or user control.

6. Input requests should indicate the type of response expected, e.g., "(Y/N)". They should request brief user responses, e.g., the numbers of options which have been displayed in a list. The amount of user typing and spelling should be minimized.

7. Keyboard input should consist only of lowercase characters (no underscores), numbers, blanks, periods, commas, and slashes, if possible. These keys are in the same positions

on most terminal keyboards. Use blanks as delimiters in preference to commas.

8. All user input should be requested through ⍞, not ⎕, to allow program control over error checking and user inputs.

9. The empty response should be allowed, to signal use of a default value or termination of execution. Defaults should be indicated in the prompting message.

10. All program dialogue with the user should be worded in impersonal terms. All appearances of "humanness", such as evoked by "I" or "YOU", should be avoided. The computer is a machine. No other impression should be given.

11. In user-prompting messages, refer to the "end-of-input" key as "RETURN" (the common typewriter label), e.g., *ADJUST PAPER ... AND PRESS ¨RETURN¨*. If the application will be used on only one terminal type, the appropriate key label may be substituted (327x=ENTER; 51xx=EXECUTE).

12. Error messages should start with ** to distinguish them from prompting messages.

13. Lengthy user input which will be needed during a repeat-execution of a command should be saved for re-use. This will reduce both the user errors in re-entry and the time needed for repeated executions.

14. User-invoked functions should detect the presence of suspended functions in the workspace and terminate execution, e.g., via:

```
    →(1=⍴⎕LC)/S1
    →0,⍴⍞←'**SUSPENDED FNS IN WS'
   S1: ....
```

This will avoid problems with shadowed names and reduced workspace size.

III. CODING CONVENTIONS AND GUIDELINES

A. CODE EFFICIENCY

Code efficiency is important, since APL programs execute with interpreter overhead. However, it is of lower priority than reliable, correct execution and maintainable, adaptable program code.

1. Programs should be coded for maximum APL execution efficiency. Vector and array operations should be used in preference to looping, whenever possible. Consult the *APL PROGRAMMING GUIDE: VECTOR OPERATIONS*, G320-6103, for examples of efficient APL statements for common operations.

2. Limit the use of the execute (⍎) function to necessary situations, e.g. converting to numbers, because it has additional execution overhead compared to normal statements. When using execute (⍎) with constructed APL statements, include comments showing the intended APL-prototype-code and describing any side effects, to give a clear understanding of the intended result.

3. Avoid combining statements artificially if execution speed is to be maximized, especially within loops.

4. Pay special attention to the efficiency of code in heavily executed loops.

III. CODING CONVENTIONS AND GUIDELINES (CONTD)

B. NAMES

1. User-invoked functions should have short names (not more than 8 chars) that are meaningful, mnemonic, and easily spelled, with no underscores.

2. Names of subfunctions and variables should be short and mnemonic, for ease in reading the code. A vector can be named *V*, a matrix *M*, an array *A*, a boolean or logical variable *B* or *L*, and an explicit result *Z*.

3. Naming Convention, to distinguish among subfunctions and local, semi-global, and global variables:

   Subfunctions - last one or two characters underscored: $SUB\underline{F}$.
   Local Variables and Labels - no underscores.
   Semi-Global Variables - first one or two characters underscored: $\underline{S}EMI$. These are variables localized only in the highest, user-invoked functions, which are used for communication among subfunctions.
   Global Variables - all characters underscored: $\underline{GLOBAL}$. These are variables which remain in the workspace after function execution has completed.
   Groups - suffix *---GP* on a descriptive name, such as the main user-invoked function: *REPORTGP*. No underscores.

4. Global variable names should have 4 or more characters. This will reduce the possibility of conflict with short local names within functions.

5. Global variables should be generated and initialized by functions. Alternatively, include an "initialization function" which creates all global variables with their initial values and documents their purpose and characteristics. This will allow recovery if a global constant or variable is accidentally destroyed.

6. All temporary variables should be localized within a function. The list of local variables should be in alphabetical order.

III. CODING CONVENTIONS AND GUIDELINES (CONTD)

B. NAMES (CONTD)

7. Local variable names of 1 or 2 characters should be re-used among all functions, to reduce symbol table requirements. Unique names should be avoided. However, mnemonic value of names should be maintained, if possible. Short local variable names will keep the header statement short and will assist in the recognition of idioms.

8. Large temporary variables should be assigned an empty value (*V*←⍳0, *V*←'', *V*←0⍴...) to recover storage after use. NOTE: ⎕*EX* can be used, but it eliminates debugging evidence that a variable has existed.

9. Use a localized temporary variable to capture unwanted temporary results, e.g., *Q*←⎕*EX* .... Do not use system variables (⎕*AI*←⎕*EX* ...). This confuses the proper use of these variables.

10. Statement labels should be short (up to 3 characters). Use comment statements to explain the processing.

11. Use a common set of labels in all functions, to reduce symbol table requirements. Avoid unique labels.

12. Labelling Convention, to distinguish internal function structure:

    S-- = *S* + numeric suffix, for downward-branch labels, unless also a loop label: *S*1.
    LP- = *LP* + numeric suffix, for upward, looping branch labels: *LP*1.
    ER- = *ER* + numeric suffix, for downward-only branch labels on error processing: *ER*1. NOTE: This is only for multiple branches to the same error processing. Error processing should immediately follow the code which detects the error. It should not be located anywhere else.

    Each label-type has a separate numeric sequence, in increasing order.

III. CODING CONVENTIONS AND GUIDELINES (CONTD)

C. BRANCHING AND LOOPING

1. For execution efficiency and function clarity, branching and looping should be used only when APL array operations are not possible or practical.

2. All branches must be to labeled lines or to 0. Do not use relative branching (→⎕LC+2), computed labels (→S5+K>0), or statement numbers. (See the label-naming convention under "NAMES".)

3. Use only branch-statement types that execute the same in either indexing origin. Acceptable branch statements usable for single and multi-way branches are:

```
→(CONDITION)/LABEL

→((COND1),(COND2),...)/S1,S2,...
```

4. Branching subfunctions (e.g. *IF*) should be used only for documentation purposes, since they usually increase execution time. They should be removed prior to execution.

5. Branch statements that exit the function should show an explicit 0.

6. Leading-decision loops should be used in preference to other types. Such loops consist of loop initialization, test for exit condition, loop processing, and unconditional branch back to the test, in that order:

```
 I←0
LP1:→(LIM<I←I+1)/S1
 ...(PROCESSING)
 →LP1
S1:...
```

7. Execution should flow downward through a function. Upward branches should occur only at the end of loops. All other branches should be directed downward in the function. This will help to produce an internal modular structure like IF-THEN-ELSE and DO-WHILE structures.

III. CODING CONVENTIONS AND GUIDELINES (CONTD)

C. BRANCHING AND LOOPING (CONTD)

8. Suspend function execution when abnormal conditions are detected, so that the error environment can be saved for debugging. An error message describing the abnormal condition should be displayed prior to termination. Use niladic branching (→) only where necessary, e.g., when a security check indicates unauthorized access.

9. Unconditional branches may be combined with preceding assignment statements using a line-separator function or the form →*LP2*,ρ*V*←..... This is useful for reducing program listing length and ⎕*CR* size.

10. Executed (⍎) branches should not be used, because execution efficiency is important.

III. CODING CONVENTIONS AND GUIDELINES (CONTD)

D. DEFINED FUNCTION AND SUBFUNCTION STRUCTURE

1. Function statements should not exceed the size of the terminal line. Preferably, function listings should fit on 8.5x11-inch paper without wrap-around, i.e., up to 65 characters per line. Exception: Header statements with many local variables.

2. Function statements should be combined only in meaningful and execution-efficient ways. Temporary storage requirements should not become excessive as a consequence of long statements.

3. Use parentheses only when necessary. No occurrences of 2 or more right parentheses are necessary. Rearrange the order of arguments to eliminate parentheses. Avoid cluttered statements with excessive or deep nests of parentheses or brackets.

4. Define subfunctions for significant operations used in 2 or more places or likely to be useful elsewhere. Use existing public library functions (e.g., *PLOT* functions) if they meet requirements.

5. Function listings including comments should fit on one page. Limit functions to 40-50 statements if possible. It may be convenient to further limit functions to the size of the terminal display, if smaller.

6. Use of global variables should be minimized. Whenever possible, data should be passed between functions as function arguments and explicit results. Alternatively, store globals in a file so that only 1 copy of them will have to be maintained for use by multiple workspaces.

7. Return codes from subfunctions, if any, should be 0 for normal completion and 1 (non-zero) for error returns.

8. Input/Output to a terminal, printer, or other I/O device should be routed to a subfunction for that type of output. This will simplify changes if a different device must be substituted.

9. Each top-level (user) function should have a common end point. This makes it easier to add extra processing code, such as calls to functions which log application usage.

III. CODING CONVENTIONS AND GUIDELINES (CONTD)

D. PROGRAM AND SUBROUTINE STRUCTURE (CONTD)

10. Prompting for user inputs should be combined into one function, if possible, to be handled all at one time.

11. Functions should be locked only when necessary. Unlocked functions allow execution to be restarted if interrupted and allow error environments to be saved for debugging.

III. CODING CONVENTIONS AND GUIDELINES (CONTD)

E. CODE AND SYSTEM DEPENDENCIES

1. No obsolete APL functions should be used: no I-beams, no semi-colon (;) terminal output.

2. Programs should not be written with system-dependent code, if possible. When necessary, isolate such code in subfunctions, so that it can be replaced easily.

3. Avoid indexing special characters or terminal control characters from ⎕*AV*. Include all printable characters as literals or variables.

4. All non-default workspace parameters (e.g., ⎕*IO*, ⎕*PW*) should be localized and set at the beginning of the affected functions.

5. Data in reports should contain only characters that are printable as both APL and EBCDIC, so that they may be printed on either a terminal or a high-speed printer with a standard print train.

6. Output formats should be device independent, if possible, within general class of device.

7. Design programs for easy modification. Avoid multiple explicit references to constants or literals. Use variables to hold "constants" that might later be changed, e.g. page size.

III. CODING CONVENTIONS AND GUIDELINES (CONTD)

F. OPTIONAL CONSIDERATIONS

The following optional guidelines may be appropriate in certain situations. Some are applicable only to unlocked APL functions.

1. Do not re-specify (destroy) function arguments within a function. This will make it possible to restart an interrupted function by branching to the first line, since all arguments will still be available for re-use. NOTE: To be able to restart, global variables also must not have been changed.

2. Make function statements restartable. Make all assignment (←) operations which update a variable the primary (leftmost, last) operation in a statement. Usually this involves isolating on separate lines any operations which increment or change a variable which existed before the statement was executed. Then, when execution of the statement is interrupted, it can be restarted without having to reset any variables. NOTE: This does not apply to assignment operations which create new variables, since these will just be recreated when a statement is restarted.

3. Write generalized subfunctions which can process arguments of variable rank and shape, and which will not fail if the argument is empty.

4. Use each local variable for only one purpose within a function. This will reduce the risk of usage-conflict errors, especially in looping code. It will also make it possible to determine whether a variable has been used yet by checking its value. For large temporary variables it may be necessary to assign an empty value to recover storage after use.

5. For functions whose main purpose is to create printed results at the terminal, write them so they yield the text to be printed as an explicit result. This will allow the display of the result to be interrupted without leaving a suspended function. It will also allow the results to be stored and saved for re-display without re-execution. A vector result with embedded "new-line" characters will print text having unequal length lines, avoiding the extra space that padding for a matrix result would require.

III. CODING CONVENTIONS AND GUIDELINES (CONTD)

F. OPTIONAL CONSIDERATIONS (CONTD)

6. Store all message text, e.g., prompting and error messages, together in separate functions or variables. This will simplify the changes necessary to convert programs for use with a different language. Be careful to avoid any code dependency on the message length.

IV. RELIABILITY AND RECOVERY CONSIDERATIONS

1. Applications should be programmed to work under all conditions and detect user input errors. User errors should not cause an APL execution error or abend.

2. The results of I/O operations should always be checked. Appropriate error processing should be executed when error conditions are detected.

3. Write an updated record back to its file as soon as the update is complete, if CPU time and cost are not critical.

4. Use replacement updating rather than additive updating to avoid double-updating if the entire job is not completed and must be rerun.

5. When processing large files, maintain a record on the file which contains the number of the last record/transaction processed, plus any other data needed for restarting execution if the job is interrupted. This can also include an indicator that processing has started. If it is found "on" when the file is opened, it indicates that processing was not complete and must be restarted.

6. Provide the capability to restart execution, either as a user-invoked function *RESTART* or as a ⎕*LX*-invoked automatic restart.

   To allow for automatic restart of execution, localize the latent expression ⎕*LX* in the highest-level function and specify it as ⎕*LX*←'→*RESTART*'. Define the function ∇*Z*←*RESTART* ... which re-establishes the necessary processing environment, e.g., shared variables with the auxiliary processor, file positioning, etc., and if successful, returns *Z*←1↓⎕*LC*. Then execution will restart whenever a saved, interrupted workspace is)*LOAD*ed. If unsuccessful, *RESTART* should display '***UNABLE TO RESTART*' and suspend execution.

V. SECURITY CONSIDERATIONS

The following list describes ways to make an APL application program secure. The level of security needed is in direct proportion to the confidentiality of the data. Items (1), (3), and (10) are very important pieces in any security scheme. Applying these techniques is the responsibility of the application developer and user.

1. All critical functions should be locked.

2. Critical functions should not call other functions which are unlocked, nor should they request data via ⎕-input.

3. All critical workspaces should be locked or RACF-protected, where available.

4. All sensitive data stored on files should be encoded and/or RACF-protected.

5. Auxiliary processor linkage and access code should be included within locked functions.

6. Encode/Decode functions should have their algorithms and/or "character string" keys changed periodically.

7. Critical variables (especially shared variables) should be localized within locked functions.

8. An additional password should be required before access is allowed to critical options within a workspace.

9. Data set names should be hidden in locked functions, or generated by a function executed by a latent expression which also checks the users sign-on number for validity.

10. Command data sets (TSIO only) should be used to control access to critical files which have been created as reserved data sets.

11. Unauthorized attempts to access data sets should be captured on access logs and/or audit trails, so that any security violators are identified.

12. An application security administrator should control or monitor items (6), (8), (10), and (11).

VI. LOCAL APL SYSTEM ENVIRONMENT CONVENTIONS

This section should be replaced with the conventions of the local APL system administration. These conventions may deal with such topics as:

1. Name and number conventions for APL workspaces, libraries, and data sets.

2. Logging of system usage by application.

3. Labelling and handling of classified programs and output.

APPENDIX A: Summary Of APL Programming Conventions

WORKSPACE DOCUMENTATION:
- *DESCRIBE* - explain use of all programs.
- ---*HOW* - explain use of 1 program.
- *CONTROL/AUTHOR* - identify owner, address, phone.

PROGRAM DOCUMENTATION:
- Comments on all functions; delete for execution.
- 1st-line summary of function processing.
- Description of arguments, results, interfaces.

HUMAN FACTORS:
- Command names up to 8 chars; no arguments.
- Long/short prompting; progress messages.
- Keep inputs short; show answers expected.
- Use ⍞-input; allow empty response.

CODING CONVENTIONS:
- Use vector/matrix/array idioms.
- Names: *SUB*$\underline{F}$; $\underline{S}$*EMI*; $\underline{GLOBAL}$; ---*GP*.
- Alphabetize local variables; use short names: *V*,*M*,*A*,*B*,*L*,*Z*
- Branching: →(*CONDITION*)/*LABEL*; downward flow.
- Labels: *S*1 (downward branch); *LP*1 (upward);
  *ER*1 (error); 0 (exit).
- Leading-decision loops.
- 40-50 lines/function; up to 65 chars/line.
- I/O through subfunctions.
- Avoid ⎕*AV* dependency; localize ⎕*IO*, ⎕*PW*.

RELIABILITY:
- Detect user-input errors.

APPENDIX B: An Example Which Follows The Conventions

This appendix contains part of the functions for an APL program. It is a comprehensive example which illustrates the application of many of the programming conventions described in this Guide. The section numbers of the specific conventions which this appendix illustrates are:

```
            I.A.5
            I.B.1, 2, 3, 4, 5
            I.C.1, 5, 6

           II.1, 2, 3, 4, 6, 8, 9, 10, 12, 14

          III.A.1, 2, 3, 4
          III.B.1, 2, 3, 6, 7, 8, 10, 11, 12
          III.C.1, 2, 3, 5, 6, 7, 9
          III.D.1, 2, 3, 4, 5, 6, 7, 10, 11
          III.E.1, 3, 4, 7
```

This example shows how a program can be meaningfully split into several functions. It makes use of a couple of generalized subfunctions which could be obtained from a library of standardized subfunctions. It is also an example of how APL processing can be composed from many basic APL idioms such as found in *APL PROGRAMMING GUIDE: VECTOR OPERATIONS*, G320-6103.

APPENDIX B: An Example Which Follows The Conventions (contd)

```
      RELABELHOW
¨RELABEL¨ RELABELS APL DEFINED FUNCTIONS WITH A COMMON SET OF
LABEL NAMES.  THIS WILL REDUCE THE NUMBER OF NAMES THE SYMBOL
TABLE MUST HOLD WHEN THESE FUNCTIONS ARE IN A WORKSPACE.

FUNCTIONS ARE RELABELLED ACCORDING TO THE FOLLOWING CONVENTION,
WHICH HELPS TO DISTINGUISH INTERNAL FUNCTION STRUCTURE:

    LP + NUMERIC SUFFIX → REPLACES ALL LABELS WHEN THE LABEL IS
          ALSO FOUND ON THE SAME LINE OR A SUBSEQUENT LINE, I.E.
          UPWARD BRANCHES.
    S + NUMERIC SUFFIX → REPLACES ALL LABELS WHICH ARE THE LAST
          OCCURRENCE OF THAT NAME IN THE FUNCTION.

LABELS UP TO 4 CHARACTERS (LP99, S999) CAN BE GENERATED.

LABELS BEGINNING WITH ¨ER¨ ARE ASSUMED TO BE LABELS ON ¨ERROR¨
PROCESSING AND ARE NOT CHANGED.  LABELS WHICH ARE NOT REFERENCED
ARE DELETED.

RELABELLING IS CANCELLED IF:

    (1) DUPLICATE LABELS ARE FOUND.
    (2) A NEW LABEL IS FOUND TO ALREADY OCCUR IN THE FUNCTION.
         THIS CAN BE CAUSED BY A BRANCH STATEMENT TO A MISSING
         LABEL.

IF THE RELABELLED FUNCTION DEFINITION CANNOT BE FIXED (⎕FX
FAILS), THE FUNCTION MATRIX WILL BE STORED UNDER THE NAME
¨BADFCN¨ WITH A NUMERIC SUFFIX, E.G., ¨BADFCN5¨.  AFTER THE
PROBLEM LINES ARE CORRECTED, THE FUNCTION CAN BE FIXED BY
¨⎕FX BADFCN5¨.  ⎕FX FAILURES WILL OCCUR IF THE RELABELLED
FUNCTION IS PENDANT OR SUSPENDED IN THE WORKSPACE.

TO USE ¨RELABEL¨, ENTER:  )PCOPY 12 UTILS RELABELGP

¨RELABEL¨ PROMPTS THE USER FOR A FUNCTION NAME OR THE NAME OF
A MATRIX OF FUNCTION NAMES.  IF AN ¨*¨ IS TYPED, THE USER WILL
BE PROMPTED TO ENTER A LIST OF NAMES.  IF A ¨3¨ IS ENTERED
INSTEAD, ALL FUNCTIONS IN THE WORKSPACE WILL BE RELABELLED.
FUNCTION NAMES ARE DISPLAYED AS THEY ARE PROCESSED.
```

APPENDIX B: An Example Which Follows The Conventions (contd)

```
      RELABEL
NAME, LISTNAME, 2,3 [2=VARS, 3=FCNS]
OR * [=ENTER LIST] :TESTFN1
TESTFN1
¨ERROR¨ LABELS UNCHANGED:  ER1 ER2
**LABEL ¨S12¨ NOT REFERENCED; DELETED.



      RELABEL
NAME, LISTNAME, 2,3 [2=VARS, 3=FCNS]
OR * [=ENTER LIST] :*
ENTER NAMES, 1 AT A TIME
DUMMY
TESTFN2

SORT LIST? (Y/N) :N
DUMMY
**NOT AN UNLOCKED FCN
TESTFN2
**NEW LABEL ¨S1¨ ALREADY OCCURS, LINES:  4 11



      )GRP RELABELGP
REQNAM̲ REPVS̲ RELREP̲ RELMAT̲ FIWRD̲ RELABEL


 1. RELABEL
 .  2. REQNAM̲
 .  2. RELMAT̲
 .  .  3. RELREP̲
 .  .  .  4. FIWRD̲
 .  .  .  4. REPVS̲


FCN SYNTAX      USE
----------      ---

I←V FIWRD̲ S     FINDS STARTING INDICES I OF WORD S IN VECTOR V
RELABEL         RE-LABELS FCN TO S1,S2,.,LP1,LP2,.(LEAVES ER.̲.)
RELMAT̲ LB       PROCESSES FUNCTION MATRIX; CALLED BY ¨RELABEL¨
Z←RELREP̲ LB     REPLACES LABELS; CALLED BY ¨RELMAT̲¨
Z←NS REPVS̲ LI   REPLACES OLD STRG (ρOS),I WITH NEW STRG NS IN VEC V̲
MZZ←REQNAM̲      PROMPTS USER FOR LIST OF NAMES
```

APPENDIX B: An Example Which Follows The Conventions (contd)

```
      ∇ RELABEL;B;C;F;L;LB;W;A̲LF;I̲;M̲;⎕IO
[1]   ⍝⍝RE-LABELS FCN TO S1,S2,.,LP1,LP2,.(LEAVES ER..)
[2]   ⍝-  2/26/81 16.12.10 TPH944
[3]   ⍝LIMITS: 4-CHAR LABELS (S999,LP99)
[4]   ⍝CALLS:  RELMAT̲, REQNAM̲
[5]   ⍝USES (SEMI)GLOBALS:  A̲LF, I̲, M̲
[6]    ⍎(1≠⍴⎕LC)/'''**SUSPENDED FNS IN WS''',⍴⎕IO←1
[7]   ⍝OBTAIN FCN NAMES
[8]    →(0∊⍴F←REQNAM̲)/0
[9]   ⍝DELETE THESE 6 UTIL FCNS FROM LIST
[10]   B← 6 7 ⍴'RELMAT̲ RELREP̲ REPVS̲  FIWRD̲  REQNAM̲ RELABEL'
[11]   B←(6,C←(1↓⍴F)⌈1↓⍴B)↑B
[12]   F←(∧/(((1↑⍴F),C)↑F)∨.≠⍉B)⌿F
[13]  ⍝PROCESS ALL NAMES IN LIST
[14]   →(0=C←1↑⍴F)/I̲←0
[15]  ⍝SPECIFY LABEL-CHARS; PUT LIKELY LABEL-CHARS AT HEAD
[16]  ⍝OF LIST, TO REDUCE SEARCH (∊) TIME
[17]   A̲LF←'SLPER1234567890⎕ABCDFGHIJKMNOQTUVWXYZ'
[18]   A̲LF←A̲LF,'∆A̲B̲C̲D̲E̲F̲G̲H̲I̲J̲K̲L̲M̲N̲O̲P̲Q̲R̲S̲T̲U̲V̲W̲X̲Y̲Z̲'
[19]  LP1:→(C<I̲←I̲+1)/0
[20]  ⍝DISPLAY NAME; CHECK WHETHER ⎕CR OBTAINED
[21]   →(~0∊⍴M̲←⎕CR ⎕←F[I̲;])/S1
[22]   →LP1,⍴⎕←'**NOT AN UNLOCKED FCN'
[23]  ⍝OBTAIN WIDTHS UP TO COLONS (WIDTHS OF LABELS)
[24]  S1:→(0=⍴W←(B←W<1↓⍴M̲)/W←+/∧\M̲≠':')/LP1
[25]  ⍝OBTAIN LINES HAVING COLONS, TO RIGHTMOST COLON
[26]   LB←M̲[B/⍳1↑⍴M̲;⍳L←⌈/W]
[27]  ⍝PAD SHORTER ¨LABELS¨ WITH BLANKS
[28]   LB←(⍴LB)⍴B\(B←,W∘.>(⍳L)-1)/,LB
[29]  ⍝DELETE ROWS WITH NON-LABEL CHARS, THEN ANY
[30]  ⍝ALL-BLANK COLS, IN CASE ¨:¨ OCCURS OTHER THAN
[31]  ⍝AFTER A LABEL (E.G., T̲H̲I̲S̲ FCN!)
[32]   LB←(∨⌿LB≠' ')/LB←(∧/LB∊' ',A̲LF)⌿LB
[33]  ⍝QUIT IF DUPLICATE LABELS
[34]   →(∧/B←∧/(B∘.≤B←⍳1↑⍴LB)∨LB∨.≠⍉LB)/S2
[35]   →LP1,⍴⎕←'**DUPLICATE LABELS:  ',,((~B)⌿LB),' '
[36]  ⍝DON'T CHANGE ¨ERROR¨ LABELS
[37]  S2:→(~∨/B←(((1↑⍴LB),2)↑LB)∧.='ER')/S3
[38]   '¨ERROR¨ LABELS UNCHANGED:  ',,(B⌿LB),' '
[39]  ⍝SKIP TO NEXT FCN IF NO LABELS LEFT
[40]  S3:→(0=1↑⍴LB←(~B)⌿LB)/LP1
[41]  ⍝REPLACE LABELS USING TEMP LABELS TO AVOID CLASHES
[42]   RELMAT̲ LB
[43]   →LP1
      ∇
```

APPENDIX B: An Example Which Follows The Conventions (contd)

```
      ∇ MZZ←REQNAM̲;IZZ;LZZ;NZZ
[1]   ⍝⍝PROMPTS USER FOR LIST OF NAMES
[2]   ⍝-  2/26/81 16.12.10 TPH944
[3]   ⍝MZZ = MATRIX OF NAMES (EMPTY VECTOR IF NO USER INPUT)
[4]   ⍝OBTAIN NAMES BEFORE ANY LOCAL VARS CREATED
[5]    NZZ←⎕NL 2 3
[6]   ⍝PROMPT FOR FCN/VAR NAME(S)
[7]    'NAME, LISTNAME, 2,3 [2=VARS, 3=FCNS]'
[8]    LZZ←ρ⍞←'OR * [=ENTER LIST] :'
[9]    →(0=ρMZZ←LZZ↓⍞)/0
[10]  ⍝¨2,3¨= USE ⎕NL 2 AND/OR 3; ¨*¨ = PROMPT FOR LIST
[11]   →('23*'=1↑MZZ)/S1,S1,S2
[12]  ⍝FORCE MATRIX SHAPE ON INPUT NAME
[13]   MZZ←(¯2↑ 1 1 ,ρMZZ)ρMZZ
[14]  ⍝EXIT IF NOT A VARIABLE (JUST A ¨NAME¨; NOT A ¨LISTNAME¨)
[15]   →(2≠⎕NC LZZ←MZZ[⎕IO;])/0
[16]  ⍝TEST FOR CHAR MATRIX (LIST OF NAMES)
[17]  ⍝IF NOT, EXIT (MUST BE SINGLE VAR-NAME)
[18]   →((' '≠0\0ρIZZ)∨2≠ρρIZZ←⍎LZZ)/0
[19]  ⍝SET CHAR MATRIX AS RESULT LIST
[20]   →S3,ρMZZ←IZZ
[21]  ⍝OBTAIN ALL FCNS AND/OR VAR NAMES IN WS
[22]  S1:MZZ←((⎕NC NZZ)∊('23'∊3↑MZZ)/ 2 3)⌿NZZ
[23]  ⍝DELETE THIS FCN FROM THE LIST. TEST FOR ANY LEFT
[24]   MZZ←(MZZ∨.≠(¯1↑ρMZZ)↑'REQNAM̲')⌿MZZ
[25]   →(0<1↑ρMZZ)/S3
[26]   →0,ρ⎕←'**NONE IN WS'
[27]  ⍝BUILD LIST FROM USER INPUT
[28]  S2:MZZ← 0 0 ρ0
[29]   'ENTER NAMES, 1 AT A TIME'
[30]  LP1:→(0=ρLZZ←,⍞)/S3
[31]  ⍝INCREASE WIDTH OF LIST OR NAME, AND CATENATE
[32]   MZZ←((0 1 ×ρLZZ)⌈ρMZZ)↑MZZ
[33]   →LP1,ρMZZ←MZZ,[⎕IO](1↓ρMZZ)↑LZZ
[34]  ⍝SKIP SORT IF 1 NAME OR EMPTY
[35]  S3:→(1≥1↑ρMZZ)/0
[36]   LZZ←ρ⍞←'SORT LIST? (Y/N) :'
[37]   →('Y'≠1↑LZZ↓⍞)/0
[38]  ⍝SORT THE LIST ALPHAB
[39]   IZZ←⎕IO+(ρMZZ)[1+⎕IO]
[40]  LP2:→((⎕IO-1)=IZZ←IZZ-1)/0
[41]   →LP2,ρMZZ←MZZ[⍋(' ',⎕AV)⍳MZZ[;IZZ];]
      ∇
```

APPENDIX B: An Example Which Follows The Conventions (contd)

```
      ∇ RELMAT̲ LB;B;D;I;L;N;C̲;V̲
[1]   ⍝⍝PROCESSES FUNCTION MATRIX; CALLED BY ¨RELABEL¨
[2]   ⍝-  2/26/81 16.12.10 TPH944
[3]   ⍝LB = MATRIX OF OLD LABELS
[4]   ⍝CALLS:  RELREP̲
[5]   ⍝USES (SEMI)GLOBALS:  C̲, I̲, M̲, V̲
[6]   ⍝RAVEL FCN WITH SEPARATORS
[7]    V̲←,(C̲←''⍴⎕AV),M̲
[8]   ⍝PERFORM LABEL REPLACEMENT; EXIT IF NO CHANGE
[9]    →(0<RELREP̲ LB)/M̲←0
[10]  ⍝REBUILD ⎕CR MATRIX, LINE-BY-LINE
[11]   M̲← 0 0 ⍴V̲←1↓V̲
[12]  LP1:→(0=B←⍴V̲)/S1
[13]  ⍝OMIT LINE IF ALL BLANKS OR EMPTY
[14]   D←∧/' '=L←(I←(V̲⍳C̲)-1)↑V̲
[15]   →(D∨B=1+⍴V̲←(I+1)↓V̲)/LP1
[16]  ⍝INCREASE MATRIX OR LINE, AND CATENATE
[17]   M̲←((0 1 ×⍴L)⌈⍴M̲)↑M̲
[18]   M̲←M̲,[1](1↓⍴M̲)↑L
[19]   →LP1
[20]  ⍝FIX FCN DEFN; CHECK WHETHER CONVERTED
[21]  S1:→(' '=0\0⍴N←⎕FX M̲)/0
[22]  ⍝IF ⎕FX FAILS, STORE FCN-MATRIX IN ¨BADFCN-¨.
[23]  ⍝EXECUTE:  BADFCN5←M̲
[24]   ⍎(B←'BADFCN',⍕I̲),'←M̲'
[25]  ⍝USER MUST CORRECT BAD ROWS; THEN:  ⎕FX BADFCN-
[26]   '**⎕FX FAILED ON ROW ',(⍕N),'.  FCN IS IN MATRIX ¨',B,'¨.'
      ∇
```

APPENDIX B: An Example Which Follows The Conventions (contd)

```
      ∇ Z←RELREP LB;B;D;J;K;L;LC;R;S;SL
   ⍝⍝REPLACES LABELS; CALLED BY "RELMAT"
   ⍝-  2/26/81 16.12.10 TPH944
   ⍝LB = MATRIX OF OLD LABELS
   ⍝Z = 0 IF FCN-MATRIX CHANGED, NEEDS ⎕FX; Z > 0 → NO CHANGE
   ⍝CALLS:  FIWRD, REPVS
   ⍝USES (SEMI)GLOBALS:  C, V
   ⍝INIT MATRIX-OF-NEW-LABELS SL, LINE-LABEL-CTR LC
    SL←(0,D←4)⍴LC←(Z←2)⍴J←0
   ⍝SUBSTITUTE TEMP LABELS "⎕--" FOR EXISTING LABELS.
   ⍝"⎕--" WILL BE VALID "NAMES" THAT WON'T CLASH WITH ANYTHING
   LP1:→((1↑⍴LB)<J←J+1)/S4
   ⍝FIND STARTING INDICES OF OLD LABEL
    →(1<⍴K←V FIWRD L←(L≠' ')/L←LB[J;])/S1
   ⍝IF ONLY 1 OCCURRENCE, MUST BE UNUSED LABEL.
   ⍝DELETE UNUSED LABEL AND COLON FROM FCN
    V←'' REPVS(1+⍴L),K
    '**LABEL "',L,'" NOT REFERENCED; DELETED.'
   ⍝SET RETURN CODE FOR ⎕FX
    →LP1,⍴Z←0
   ⍝IS LAST OCCURRENCE THE LABEL (CHECK FOR COLON)?
   S1:B←1+':'=V[(⍴V)⌊(¯1↑K)+⍴L]
   ⍝CHECK FOR NO BRANCH IF LAST 2 OCCURRENCES IN SAME LINE
   ⍝(LABEL REFERENCE WITHIN SAME LINE, IF NO "→", = S--)
    →(C∨.=S←K[¯1+⍴K]↓K[⍴K]↑V)/S2
    B←1+'→'∧.≠S
   ⍝IF NOT, IS LOOP LABEL.  INCREM COUNTER OF THAT TYPE AND
   ⍝BUILD LOOP OR FORWARD BRANCH LABEL, AS APPROP
   S2:S←(2 ¯1[B]↑'LP S'),⍕LC[B]←LC[B]+1
   ⍝IF SAME AS OLD LABEL, SKIP TO NEXT LABEL
    →((⍴L)≠⍴S)/S3
    →(L∧.=S)/LP1
   ⍝ADD TO LIST OF NEW LABELS
   S3:SL←SL,[1] D↑S
   ⍝REPLACE OLD LABEL WITH TEMP LABEL
    →LP1,⍴V←('⎕',S) REPVS(⍴L),K
   ⍝QUIT IF NO NEW LABELS
   S4:→(0=R←1↑⍴SL)/J←0
   ⍝REPLACE TEMP LABELS WITH NEW LABELS
   LP2:→(R<J←J+1)/Z←0
   ⍝CHECK FOR OCCURRENCE OF NEW LABEL; QUIT IF EXISTS
   ⍝(MAY HAVE BEEN A MISSING LABEL IN FCN)
    →(0=⍴K←V FIWRD S←(S≠' ')/S←SL[J;])/S5
   ⍝COMPUTE LINE NUMBERS WHERE FOUND; DISPLAY MSG; Z←NO ⎕FX
    Z←⍴,K←¯1++/K∘.>(V=C)/⍳⍴V
    →0,⍴⎕←'**NEW LABEL "',S,'" ALREADY OCCURS, LINES:  ',⍕K
   ⍝FIND STARTING INDICES OF TEMP LABEL
   S5:→(0=⍴K←V FIWRD L←(L≠' ')/L←'⎕',SL[J;])/LP2
   ⍝REPLACE WITH NEW LABEL
    →LP2,⍴V←S REPVS(⍴L),K
      ∇
```

APL Programming Guide
Programming Conventions

G320-6735-0

READER'S COMMENT FORM

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Possible topics for comment are:

Clarity   Accuracy   Completeness   Organization   Coding   Retrieval   Legibility

If you wish a reply, give your name, company, mailing address, and date:

____

____

____

____

What is your occupation? ____________________

Number of latest Newsletter associated with this publication: ____________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

G320-6735-0

Reader's Comment Form

Cut or Fold Along Line

Fold and tape Please Do Not Staple Fold and tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department 824
1133 Westchester Avenue
White Plains, New York 10604

Fold and tape Please Do Not Staple Fold and tape

IBM®

APL Programming Guide
Programming Conventions

G320-6735-0

READER'S COMMENT FORM

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Possible topics for comment are:

Clarity   Accuracy   Completeness   Organization   Coding   Retrieval   Legibility

If you wish a reply, give your name, company, mailing address, and date:

____

____

____

____

What is your occupation? ______________________________

Number of latest Newsletter associated with this publication: ____________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

G320-6735-0

Reader's Comment Form

Cut or Fold Along Line

Fold and tape Please Do Not Staple Fold and tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department 824
1133 Westchester Avenue
White Plains, New York 10604

Fold and tape Please Do Not Staple Fold and tape

IBM®

APL Programming Guide Programming Conventions Printed in U.S.A. G320-6735-0

G320-6735-0

APL Programming Guide Programming Conventions Printed in U.S.A. G320-6735-0

IBM

G320-6735-00